7th UCAAT User Conference on Advanced Automated Testing

ETSI

AUTOMATING EXPLORATORY TESTING

Presented by Andreas Ulrich, Siemens AG

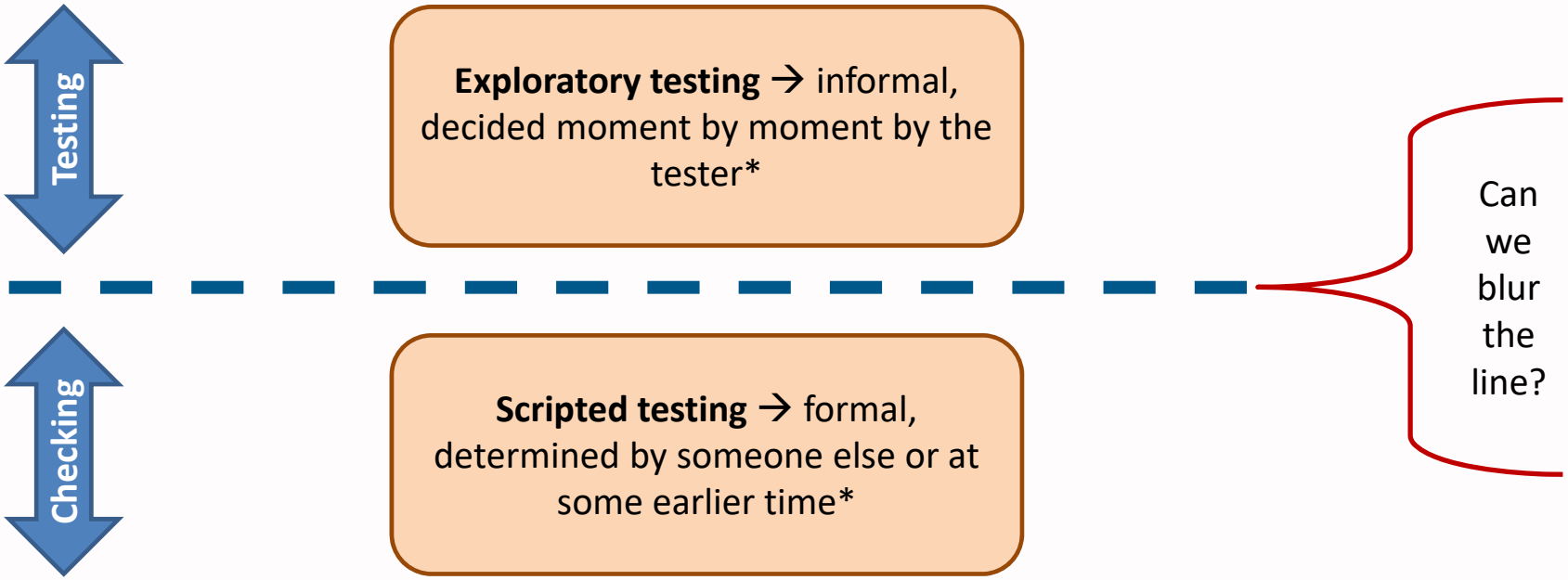
SIEMENS
Ingenuity for life

Exploratory Testing

- James Bach: “Testing is an exploratory process. It’s not just sometimes exploratory; it is inherently exploratory.”

- **Exploratory Testing** is defined as simultaneous
  - Learning,
  - Test design and
  - Test execution.

Testing vs Checking



* James Bach, https://www.satisfice.com/exploratory-testing

Adaptive Testing

- Testing of self-contained software component
- Only the component API is accessed → black-box test
- Scenario tests
  - Structured sequence of predefined **test commands**
- Tester reacts to SUT responses at runtime
  - Selection of next test command according to an **overall goal**

Adaptive testing approach

```
Given: Set of Test Commands
var tcmd = Reset();
while(true)
{
   tcmd.Invoke();
   tcmd = SelectNext();
}
```

Test Commands

- A test command = SUT interaction & local state update

  Follows the "4-As" pattern; extension of "3-As" for unit testing
  - **Arrange**: prepare input parameters for SUT call
  - **Act**: perform SUT call
  - **Assert**: validate correctness of returned data
  - **Adapt**: update local state in tester

- A test command is conditioned
  - A **condition** describes the state that enables the test command
  - Finding the right condition and state representation is a **creative, exploratory act**

Example: Stack

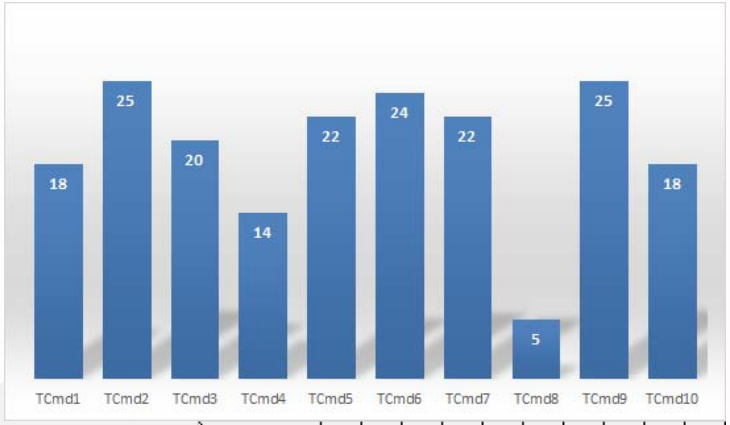
```
[Condition(true)] TPush()
{ sut.Push(x);
  assert(sut.Length, i+1); i++; }

[Condition(i > 0)] TPeek()
{ sut.Peek();
  assert(sut.Length, i) }

[Condition(i > 0)] TPop()
{ x = sut.Pop();
  assert(sut.Length, i-1); i--; }
```

Coverage as Overall Goal

- When is a test command covered?
  - When it was executed.

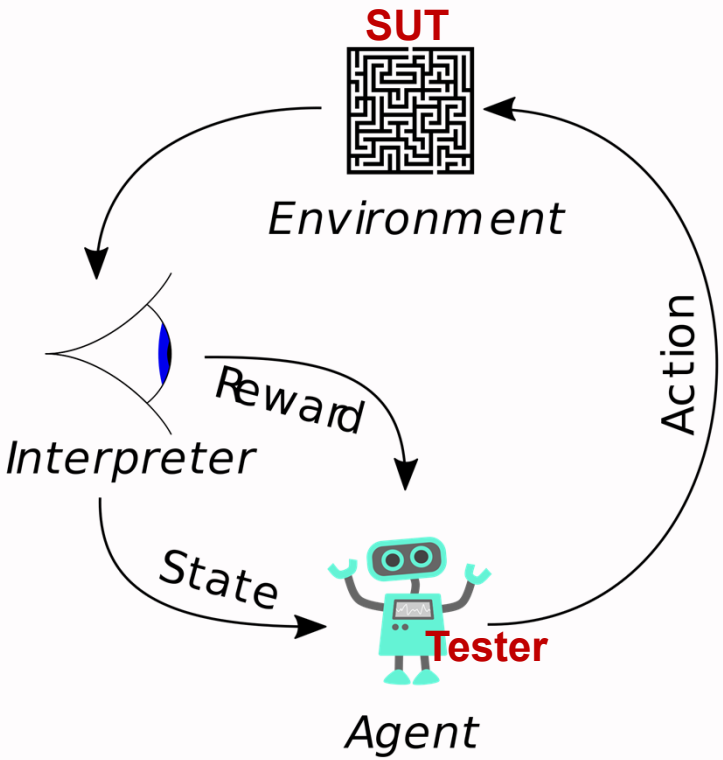


- When is the set of test commands covered?
  - Syntactical coverage over test command definitions
    - Counting test commands, pairs of test commands etc.
  - Semantical coverage over states reached in test execution
    - Counting states

	TCmd1	TCmd2	TCmd3	TCmd4	TCmd5	TCmd6	TCmd7	TCmd8	TCmd9	TCmd10
TCmd1	1	2	1	0	1	1	1	1	1	1
TCmd2	2	1	2	1	2	2	1	1	2	2
TCmd3	2	1	1	1	1	1	1	0	1	1
TCmd4	1	1	1	1	1	1	0	0	1	1
TCmd5	1	1	1	1	1	2	1	0	2	1
TCmd6	1	2	2	1	1	2	2	0	2	1
TCmd7	1	1	1	1	1	1	1	1	2	1
TCmd8	1	1	0	0	0	0	1	0	1	0
TCmd9	1	2	1	1	2	2	2	0	1	1
TCmd10	1	1	2	0	2	1	1	0	1	1

Reinforcement Learning

- Elements in reinforcement learning
  - **State**: Set of test commands enabled in current location
  - **Action**: Next test command to be executed
  - **Reward function**: Maximise a chosen coverage criterion
- Selection of next test command is a stochastic process, which results into SUT exploration
- **Issue**: Recognise states of SUT, which is black-box
  - Use approximations
  - Learn states from performed test sequences



(figure source: wikipedia.org, adapted)

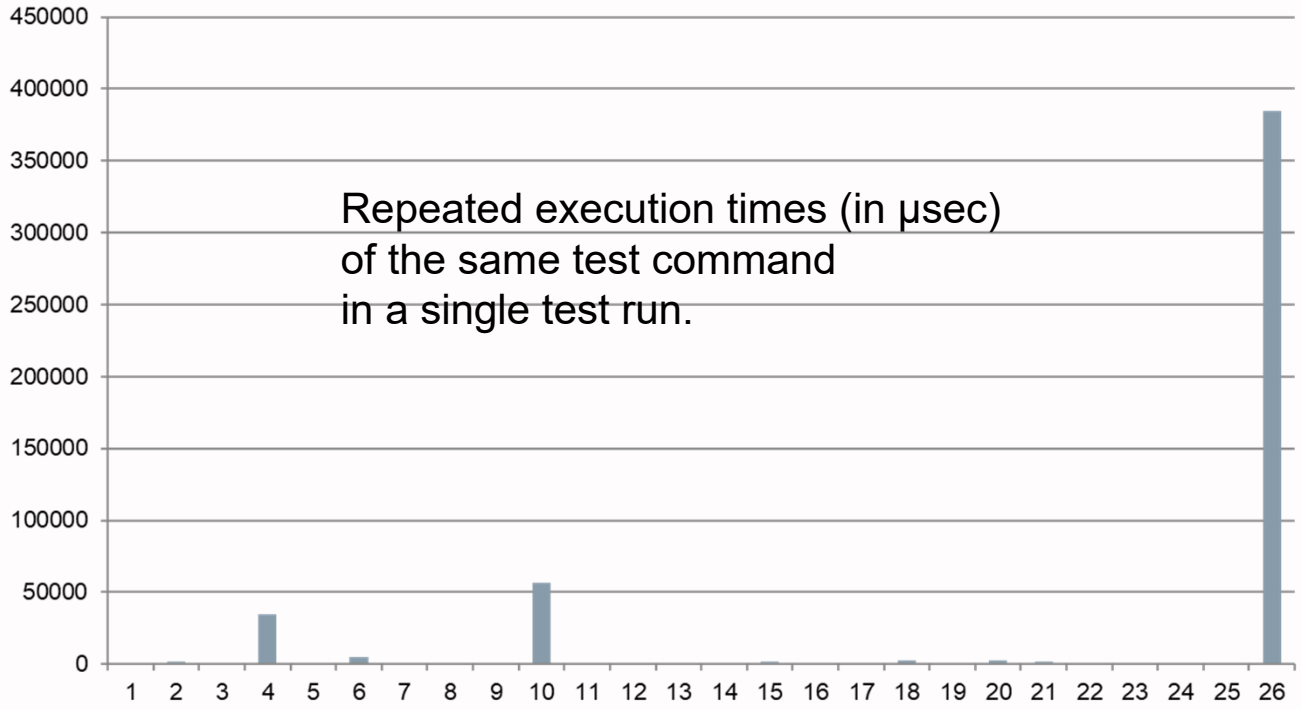
What Difference Does It Make – Example 1: Deep State Failures

- 5 adaptive tests detected a failure that 3453 unit tests and 1084 hard-coded integration tests were unable to find!
- Adaptive tests remain effective in finding failures over the entire development cycle

Test	Date/Time	Duration	Duration Average	Passed	Failed Fixtures	Failed	Run
Adaptive Tests	2017-09-20 03:15:14	8m 3.523s	1m 20s 587ms	1	0	5	6
Unit Tests	2017-09-20 03:07:09	1h 7m 44.525s	1s 177ms	3453	0	0	3453
Integration Tests	2017-09-20 01:59:19	1h 42m 54.007s	5s 696ms	1084	0	0	1084
	Summary	2h 58m 42.055s	2s 360ms	4538	0	5	4543

What Difference Does It Make – Example 2: Performance Degradation

- Tracking of the execution time of test commands
  - Within the same test run
    → Reliability tests
  - Over multiple test runs over time
    → Regression tests
- Expectation on execution times
  - Constant
  - Proportional
  - Learned



Conclusions

- Exploratory testing doesn't need to stop with scripted tests
- Adaptive testing preserves the exploration capability during runtime
- Reinforcement learning helps find optimal solutions to a given coverage goal utilising:
  - Exploration of potentially new SUT behaviour
  - Exploitation of learned knowledge